UNIT 1:
*WHAT IS DATA STRUCTURE.
A) data structure is a way to store and organize data so that it can be used efficiently. there are 2 types of data structure.
1. PRIMITIVE DATA STRUCTURE: any keywords that is found in datatype is called primitive data structure.(ex: int, float, double, char, pointer etc.).they are also known as reserved words.
2. NON PRIMITIVE DATA STRUCTURE: these are the data structure in which we perform all the major operations like sorting, merging, and many more. these are if two types.
LINEAR: a data structure is called linear data structure if all its elements are arranged in the linear order. there are 4 types of linear data structure.
*array: it is a collection of similar data types. for ex: if we want to store the roll no. of 10 students instead of creating 10 int type variable we can store that in array so that can reduce length of code and save our memory.
*stack: in stack data structure elements are stored in LIFO(last in first out) order means data added in last will be removed first. addition of elements in an stack is known as push operation and deletion of element is known as pop operation.
*queue: in queue data structure elements are stored in a FIFO(first in first out) order means data added first will removed first. insertion of data is known as enqueue and deletion of data elelments is known as dequeue.
*linked list: in linked list data structure, data elements are connected through a series of nodes. each nodes contain the data and address to the next node.
NON-LINEAR: elements in non linear data structure are not in any sequence. instead they are arranged in a hierarchical manner where one element will be connected to one or more elements. non linear data structure is divided into 2 types.
*trees: it is one type of non linear data structure. it is a collection of nodes and edges. there will be unique node called root in tree. tree is a hierarchical model. it always contain n-1 edges.
*graph: it is one type of non linear data structure. it is collection of vetices and edges. there will be no unique node called root in graph. graph is a network model. it contain no. of edges depending upon graph.

*ABSTRACT DATA TYPE:
A) abstract data type is a mathematical model that logically represents a datatype. abstract datatype only tells you the essential features without including background details. it specifies set of data and set of operation that can be performed on the data. this is also called information hiding. it is a logical description of how we view data and operation without respective of how they will be implemented. the implementation of abstract data type, often referred as a data type.

*SPARSE MATRICES.
A) it is a two dimensional data object which is made by m rows and n columns, so that we can say the number of data value in spares matrix are m and n. in any matrix, the element have 0 value are more than the others elements called a sparse matrix.
ADVANTAGES:
*storage: as we know sparse matrix contain lesser non zero elements than zero so less memory can be used to store elements. it evaluates only the non zero elements.
*computing time: computing time can be saved by logically designing ds traversing

only non zero elements.
SPARSE MATRIX REPRESENTATION: the non zero elements can be stored with triples. i.e; row, column, and value. it can be representated in following way.
1. array representation: the 2d array can be used to represents a sparse matrix there are three rows named as:
*row: it is an index of a row where a non zero elements is located.
*column: it is an index of a column where non zero elements is located.
*value: the value of a non zero elements is located at the index.(row, column)
2. linked list representation: it is a data structure used to represents a sparse matrix. each nodes has 4 fields where as array have 3 fields.
*row: it is an index of a row where a non zero elements is located.
*column: it is an index of a column where a non zero elements is located.
*value: it is value of non zero element located at index.
*nodes: it store address of next node.

*EXPLAIN THE REPRESENTATION OF STRING WITH AN EXAMPLE.
A string is a sequence of characters. It can include letters, numbers, symbols, and spaces. For example, "hello", "1234", and "Hi there!" are all strings.
How Strings are Represented in Data Structures:
In languages like C, strings are represented as arrays of characters with a special character ('\0') at the end to indicate the end of the string.
syntax:
char string[] = "hello";
// Internally, it is stored as ['h', 'e', 'l', 'l', 'o', '\0']. String is an immutable object, which means the value of the string cannot be changed. In string array only fixed set of elements can be stored to stop it is an index based data structure, which starts from the 0th position the 1st element will take place in index zero and the second element would take place in index one, and so on.

*REPRESENTATION OF ARRAY WITH AN EXAMPLE.
A) the collection of similar data types is known as an array.
   *it is a derived data type which is constructed with help of primitive data type.
   *it store same types of data types in a contiguous memory.
   *it can't contain dis-similar types of data.
   *array index always start with zero.
  there are two types of arrays.
1. single dimensional array:
* it is known as one dimensional array.
* it use only one subscript({}) to define the elelments of array.
* declaration: data-type var-name [values].
syntax:
int mynum[9]={1,2,3,4,5,6,7,8,9};
char a[5]= {'A' 'B' 'C' 'D' 'E'};
2. multi-dimensional array:
*it uses more then one one subscript to describe the array element.
there is another topic in two dimensional array i.e; two dimensional array have 2 subscript. one subscript is used to represent row and another subscript represents column;
*declaration of two dimensional array: data-type var-name [row] [column].
syntax:

```
string letters[2][4] = {
  { "A", "B", "C", "D" },
  { "E", "F", "G", "H" }
};.
```

*ARRAY ADVANTAGE AND DISADVANTAGE.
A) ADVANTAGE:
   * we can store n no. of elements.
   * we can store the elements all present on same data type.
   * we can implement stack, queue, linked list etc.
   DISADVANTAGE:
   * memory wastage is present.
   *implementation takes lot of time.

*DEFINE MATRICES AND ITS TYPES WITH AN EXAMPLE.
A) * it is a two dimensional data structure.
   * all of its elements are of same type.
   * a data frame is two dimensional and diff column may contain diff data types.
   * all column must have the same length. and there are mainly two types of matrices
SPECIAL MATRICES MARTRIX: the various types of matrices are row matrix column matrix null matrix square matrix diagonal matrix upper triangular matrix lower triangular matrix symmetric matrix and anti symmetric matrix. The special type of matrices are
1. Triangular matrix: in mathematics and computer science, a triangular matrix is a special kind of square matrix where ll non 0 value gather at only one side of any diagonal of the matrix. There are two types of triangular matrix
* lower triangular martrix: a square matrix is called over triangular if all the entries above the main diagnosis are zero.
* upper triangular matrix: square matrix is called upper triangular if all the entries below the main diagonal are zero.
2. Symmetric matrix: symmetric matrix is a square matrix which is equal to its transpose. It's symmetric matrix is always the square matrix.
3. Diagonal matrix: attack on matrix is defined as a square matrix in which all of diagonal entries are zero.
SPARES MATRICES: it is a two dimensional data object which is made by m rows and n columns, so that we can say the number of data value in spares matrix are m and n. in any matrix, the element have 0 value are more than the others elements called a sparse matrix.

UNIT 2:
*ILLUSTRATE STACK OPERATION WITH AN EXAMPLE.
A)* stack is a non primitive data structure.
  * stack is a container of objects that are inserted and removed according to the last in first out principle.
  * that has only one end whereas the queue has two ends (front and rear).
  operation on stack:
  *push(): when we insert an element in a stack then the operation is known as push.
  *pop(): when we delete an element from the stack the operation is known as a pop.
  *isempty(): it determines whether the stack is empty or not.
  *isfull(): it determines whether the stack is full or not.

*peek(): the queue and stack function allows you to peer into your stack without deleting any item.
*change(): it changes the element at the given position.
*display(): it prints all the elements available in the stack.

*EXPLAIN OPERATOR PRESEIDENCE RULE TO EVALUATE AN EXPRESSION.
A) precedence: this rule keeps the information about which operator has to be performed first when the expression contains more than one operator.
associative: the order to be followed when the expression contains two or more operators with an equal precedence. It can be right to left or left to right.

*COMPARE DOUBLE ENDED QUEUE ABD CIRCULAR QUEUE WITH AN EXAMPLE.
A) DEQUEU:
* you can insert and remove elements from both the front and the back.
* Can be implemented using arrays or linked lists.
* Flexible due to operations at both ends.
* it does not follow the fifo principle.
CIRCULAR QUEUE:
* circular queue except the last element of the queue is connected to the first element.
* insertion happens at the rear and the deletion happens from the front.
* efficient use of memory because it reuse empty slots.
* typically implemented using an array with wrap around logic.

*DISCUSS ABOUT THE APPLICATIONS OF QUEUE.
A) a queue is a data structure in which whatever comes first will go out first. it follows the fifo policy. In queue the insertion is done from one end known as the rear end whereas the deletion is done from another and known as the front.
APPLICATION OF QUEUE:
* when you send multiple documents to a printer, they are lined up in a queue and the first is document sent to the printer is printed first.
*In operating systems task waiting to be executed or lined up in a queue the task that was added first is executed first.
*Call center use queues to manage incoming calls the first caller to enter the queue is the first to be served.

*EXPLAIN OPERATION ON QUEUE.
A) a queue is a data structure in which whatever comes first will go out first. it follows the fifo policy. In queue the insertion is done from one end known as the rear end whereas the deletion is done from another and known as the front.
OPERATIONS:
*enqueue: the enqueue operation is used to insert the element at the rear end of the queue.
*dequeue: the Dequeue operation performed the deletion from the front end of the queue.
*peek: it is the operation that returns the element but does not delete it.
*queue overflow(isfull): when the queue is completely full, then it shows the overflow condition.
* queue underflow(isempty): when the queue is empty i.e; no element are in the queue then it shows the underflow condition.

*LINKED REPRESENTATION OF STACK.
A) A stack is a data structure that follows the "Last In, First Out" (LIFO) principle, meaning the last item added is the first one to be removed. When using a linked representation of a stack, we use a linked list to manage the elements in the stack. Here's an easy explanation:

1. **Nodes**: In a linked list, each element is called a node. Each node contains two parts: the data (the value) and a reference (or pointer) to the next node in the list.

2. **Top Pointer**: In a stack, we keep track of the top element with a special pointer called the top. This pointer always points to the most recently added node, which is at the top of the stack.

3. **Push Operation**: When you add (push) an element to the stack:
   - A new node is created.
   - The new node's reference points to the current top node.
   - The top pointer is updated to point to this new node.

4. **Pop Operation**: When you remove (pop) an element from the stack:
   - The node at the top is removed.
   - The top pointer is updated to point to the next node in the list, which was referenced by the removed node.

5. **Empty Stack**: When the stack is empty, the top pointer is set to `null` (or `None` in some programming languages), indicating that there are no nodes in the stack.

Using a linked list for a stack is efficient because it allows dynamic resizing and ensures constant time complexity (O(1)) for both push and pop operations, regardless of the number of elements in the stack.

*ARRAY REPRESENTATION IN QUEUE.
A) A queue is a data structure that works on a "First In, First Out" (FIFO) basis, meaning the first element added is the first one removed. When using an array to represent a queue, we store the elements in a linear order inside an array. Here’s an easy explanation of how this works:

1. **Array**: Think of an array as a row of boxes where each box can hold one element of the queue.

2. **Front and Rear Pointers**: We use two pointers (or indices) to keep track of where the front and the rear of the queue are.
   - **Front**: Points to the first element in the queue (the element to be removed next).
   - **Rear**: Points to the last element added to the queue.

3. **Enqueue Operation**: Adding an element to the queue.
   - Place the new element in the array position indicated by the rear pointer.

- Move the rear pointer to the next position.
- If the rear pointer reaches the end of the array, it wraps around to the beginning (in a circular queue).

4. **Dequeue Operation**: Removing an element from the queue.
- Remove the element from the array position indicated by the front pointer.
- Move the front pointer to the next position.
- If the front pointer reaches the end of the array, it wraps around to the beginning (in a circular queue).

5. **Circular Queue**: To efficiently use the array space, we use a circular queue. This means that when we reach the end of the array, we go back to the beginning if there is space available. It avoids the problem of running out of space when there are still unused spots at the beginning of the array.
Using an array for a queue is simple and provides quick access to elements, but you need to handle the situation when the array is full and manage the circular nature to use space efficiently.

*EXPLAIN BRIEFLY ABOUT CIRCULAR QUEUE.
A) * in circular queue, all the nodes are represented as a circular.
*It is similar to linear Queue expect that the last element of Queue is connected to the first element.
*it is also known as ring buffer as all ends are connected to the another end.
* the drawback that occurs in a linear queue is overcome by using the circular queue.
* if the empty spaces is available in circular queue, the new element can be added in an empty space.

UNIT 3:
*EXPLAIN ABOUT SINGLE LINKED LIST WITH AN EXAMPLE.
A) linked list: this is a very common data structure which consists of groups of nodes in a sequence. each node holds its own data and the address of the next node hence forming a chain like structure.
*single linked list: It contains nodes which have data and address part which points to the next node.
The operation we can perform on single linked list are
insertion: insertion into a single linked list can be performed at different position stop based on the position of the new node being inserted, the insertion is categorized. Insertion at beginning, insertion at the end of the list, insertion after specified node.
deletion: the deletion of a note from a singly linked list can be performed at different position based on the position of the node being deleted the operation is categorized into. Deletion at beginning, deletion at the end of the list, deletion after specified not, traversing, searching.

*EXPLAIN ABOUT CIRCULAR LIST IN DETAILS .
A) A circular list is a type of linked list where the last node points back to the first node, creating a circle. Unlike a regular linked list, which ends at a node with a null reference, a circular list never ends—it loops back to the beginning. Here's a detailed yet simple explanation:

Structure of a Circular List

1. **Nodes**: Just like a regular linked list, a circular list consists of nodes. Each node has two parts:
   - **Data**: The value stored in the node.
   - **Next**: A reference (or pointer) to the next node in the list.

2. **Head Node**: The starting point of the list. In a circular list, the head node is also linked to by the last node, forming a circle.

Characteristics of a Circular List

- **No Null References**: Unlike a regular linked list where the last node points to null, in a circular list, the last node points back to the head node.
- **Traversing**: You can start at any node and follow the next pointers to traverse the entire list, eventually returning to the starting node.
- **Continuous Loop**: There's no natural end to the list—you'll keep looping around if you continue traversing.

Operations in a Circular List

1. **Traversal**:
   - Start at the head node.
   - Follow the next pointers to visit each node.
   - Stop when you reach the starting node again.

2. **Insertion**:
   - **At the Beginning**:
     - Create a new node.
     - Point its next reference to the current head node.
     - Update the last node's next reference to this new node.
     - Update the head to this new node.
   - **At the End**:
     - Create a new node.
     - Point the last node's next reference to this new node.
     - Point the new node's next reference to the head node.
     - Update the last node to this new node.
   - **In the Middle**:
     - Create a new node.
     - Point the new node’s next reference to the next node of the current position.
     - Update the current node’s next reference to this new node.

3. **Deletion**:
   - **From the Beginning**:
     - Point the last node’s next reference to the node after the head.
     - Update the head to the next node.
   - **From the End**:
     - Traverse the list to find the second-last node.
     - Point the second-last node’s next reference to the head node.

    - Update the last node.
  - **From the Middle**:
    - Traverse the list to find the node just before the one to be deleted.
    - Point its next reference to the node after the one to be deleted.

*DESCRRIBE ABOUT HASH FUNCTION.
A) A hash function in data structures is a special process that takes an input (called a key) and turns it into a unique number (called a hash value or hash code). This number is then used to quickly find or store the data in a table called a hash table.
TYPES OF HASH FUNCTION:
*division method: It involves taking the key and dividing it by the size of the hash table, then using the remainder as the hash value.
*mid square method: the key is divided into separate parts.i after dividing the parts combine this parts by adding it.

*WHAT IS STATICS HASHING FUNCTION.
A) Static hashing is a technique used in data structures to map data keys to specific locations in a fixed-size table, called a hash table, using a hash function. In simple terms, it's a way to organize and access data efficiently with a hash function, but the size of the table doesn't change after it's created.
Example
Hash Table Size: 10 slots.
Hash Function: hash(key) = key % 10.
Data to Insert: Keys 15, 25, 35.
Inserting Data:

For key 15: hash(15) = 15 % 10 = 5. Store data in slot 5.
For key 25: hash(25) = 25 % 10 = 5. Slot 5 is occupied, so handle the collision (e.g., by chaining).
For key 35: hash(35) = 35 % 10 = 5. Slot 5 is occupied, so handle the collision.
Retrieving Data:

To find the data for key 15, compute hash(15) = 15 % 10 = 5 and look in slot 5.
Collision Handling:
Since the table size is fixed, collisions can occur when different keys produce the same hash value. Static hashing must handle these collisions, usually with:

Chaining: Each slot in the hash table points to a linked list of all elements that hash to the same slot.
Example: Slot 5 would have a list containing keys 15, 25, and 35.
Open Addressing: When a collision occurs, find another open slot within the table to store the data.

*DIFFERNCE BW DOUBLE LINKED LIST AND CIRCULAR LIST.
A) double linked list:
   * each node has three parts i.e; data, a pointer to the next node, and a pointer to the previous node.
   * you can start at any node and move forward to the end or backward to the beginning.

* Requires more memory per node due to the extra pointer for the previous node.
circular linked list:
* Each node has two parts: data and a pointer to the next node.
* You can only move in one direction unless you make variation.
* Typically uses less memory than a doubly linked list because it does not store previous pointer.

*DISCUSS IN DETAIL ABOUT OVERFLOW HANDLING.
A) Overflow handling in data structures refers to how the system manages situations where the data or operations exceed the allocated capacity.
*collision in hashing:
Collision in hashing occurs when two different keys generate the same hash value or index in a hash table.
*Collision Handling:
Since the table size is fixed, collisions can occur when different keys produce the same hash value. Static hashing must handle these collisions, usually with:

Chaining: Each slot in the hash table points to a linked list of all elements that hash to the same slot.
Example: Slot 5 would have a list containing keys 15, 25, and 35.
Open Addressing: When a collision occurs, find another open slot within the table to store the data.

*ILLUSTRATE DOUBLE LINKED LIST WITH AN EXAMPLE.
A) each node has three parts i.e; data, a pointer to the next node, and a pointer to the previous node.
* you can start at any node and move forward to the end or backward to the beginning.
* Requires more memory per node due to the extra pointer for the previous node.
EXAMPLE: WRITE AN EXAMPLE: